Practice Midterm 2

CS106B, Fall 2022

(Print name legibly)

(SUID **number**)

Exam Instructions

There are **5** questions worth a total of **65** points. Write all answers directly on the exam paper. This printed exam is closed-book and closed-device; you may refer only to our provided reference sheet. You are required to write your SUID number in the blank at the top of each odd-numbered page.

C++ Coding Guidelines

Unless otherwise restricted in the instructors for a specific problem, you are free to use any of the CS106B libraries and classes. You don't need *#include* statements in your solutions; just assume the required header files (*vector.h*, *strlib.h*, etc.) are visible. You do not need to declare prototypes. You are free to create helper functions unless the problem states otherwise. Comments are not required, but when your code is incorrect, comments could clarify your intentions and help the graders award partial credit.

The Stanford University Honor Code (*This version of the Honor Code is now outdated.*)

A. The Honor Code is an undertaking of the students, individually and collectively:

(1) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;

(2) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

B. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid as far as practicable, academic procedures that create temptations to violate the Honor Code.

C. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

In signing below, I acknowledge, accept, and agree to abide by the Honor Code.

(signature)

This cover sheet has been revised to reflect the format you should expect to see on this quarter's exam.

This page intentionally left blank.
You may use this space for scratch work (it will not be graded).

Problem 1: C++ fundamentals (12 points)

The **`trimPrefix`** function takes one argument, a string passed by reference, and modifies the string to remove all non-alphabetic characters from the string's prefix. As an example, trimming **`"98% Fresh!!"`** removes the first four characters changing the string into "**`Fresh!!`**".

Write the **`trimPrefix`** function. You may use any string functions from the C++ standard and/or Stanford libraries.

```
void trimPrefix(string& str)
```

A **`PROVIDED_TEST`** confirms the example **`"98% Fresh!!"`**. Complete the **`STUDENT_TEST`** below with two test cases on additional inputs. For each input, briefly explain why you chose it and how its coverage is distinct from the provided test and your other test.

```
STUDENT_TEST("My test cases for trimPrefix") {
```

Problem 2: ADTs (12 points)
Your dorm is holding a campus scavenger hunt! Each team is assigned a set of items to find and snap a picture as proof. There is a campus map where each key is a place on campus associated with the items that can be found there. Here is an example **`campus`** and **`huntlist:`**

```
Map<string, Set<string>> campus =
                 { "bookstore" : { "book", "cafe", "postcard" },
                   "library"   : { "bench", "book", "fountain" },
                   "museum"    : { "bench", "cafe", "sculpture" },
                   "quad"      : { "arch", "bench", "sculpture" },
                   "stadium"   : { "bench", "jumbotron" }  }

Set<string> huntList =  { "arch", "bench", "fountain", "sculpture" }
```

Write the **`collect`** function:

```
Map<string, string> collect(Set<string> huntList,
                            Map<string, Set<string>>& campus)
```

This function identifies which places to visit and returns a result map associating each found item to the place it was found.

Here is the required algorithm for **`collect`**:

1. Start with an empty result map
2. Iterate over each place in the campus map
3. If this place has **2 or more items remaining on your hunt list**:
   a. Visit it and cross off these items off your list
   b. Add each of these items to the result map associated with the place it was found. *Hint*: the higher-level set operations will be handy for tasks in Step 3.
4. Continue until you have found all items or you run out of places to consider
5. Return the result map

Note that you are not trying to visit the optimally fewest places to collect the most items, you are to follow the algorithm exactly as described.

The call **`collect(huntList, campus)`** returns this result map:

```
result = { "arch"      : "quad",
           "bench"     : "library",
           "fountain"  : "library",
           "sculpture" : "quad"     }
```

You don't visit the bookstore since it has none of your items. You do visit the library and collect both **`bench`** and **`fountain`**. You don't visit the museum since it has only one of your remaining items. You do visit the quad to collect both **`arch`** and **`sculpture`**. Now that all items have been collected, you do not consider visiting any further places.

```
Map<string, string> collect(Set<string> huntList,
                            Map<string, Set<string>>& campus)
```

Problem 3: Code study: ADTs and Big-O (12 points)
The "cleave" operation moves **k** elements from one collection to another. You are given four versions of cleave; one each for the collection types Vector, Set, Stack, and Queue.

```
void cleaveVector(Vector<int>& one,
           Vector<int>& two, int k)
{
    for (int i = 0; i < k; i++) {
        two.insert(0, one[0]);
        one.remove(0);
    }
}
```

```
void cleaveSet(Set<int>& one,
           Set<int>& two, int k)
{
    for (int i = 0; i < k; i++) {
        two.add(one.first());
        one.remove(one.first());
    }
}
```

```
void cleaveStack(Stack<int>& one,
           Stack<int>& two, int k)
{
    for (int i = 0; i < k; i++) {
        two.push(one.pop());
    }
}
```

```
void cleaveQueue(Queue<int>& one,
           Queue<int>& two, int k)
{
    for (int i = 0; i < k; i++) {
        two.enqueue(one.dequeue());
    }
}
```

Show the contents of **`a`** and **`b`** after each call to cleave.

```
Vector<int> a = {9, 3, 6, 1, 5};
Vector<int> b;
cleaveVector(a, b, 2);
```

```
a =
b =
```

```
Set<int> a = {9, 3, 6, 1, 5};
Set<int> b;
cleaveSet(a, b, 2);
```

```
a =
b =
```

```
Stack<int> a = {9, 3, 6, 1, 5};
Stack<int> b;
cleaveStack(a, b, 2);
```

```
a =
b =
```

```
Queue<int> a = {9, 3, 6, 1, 5};
Queue<int> b;
cleaveQueue(a, b, 2);
```

```
a =
b =
```

Reminder: ***Stack*** *elements are listed in order* ***bottom to top****.* ***Queue*** *elements listed in order* ***front to back****.*

Give the Big-O runtime of each call to cleave in terms of **`N`** where **`N`** = **`a.size()`** and **`b`** is empty.

`cleaveVector(a, b, 2)`		`cleaveVector(a, b, a.size()/2)`	
`cleaveSet(a, b, 2)`		`cleaveSet(a, b, a.size()/2)`	
`cleaveStack(a, b, 2)`		`cleaveStack(a, b, a.size()/2)`	
`cleaveQueue(a, b, 2)`		`cleaveQueue(a, b, a.size()/2)`	

Problem 4: Recursion (14 points)
There are **2^n** possible sequences that can result from flipping a fair coin **n** times. We define a **kGood** sequence as one that never has more than **k** repeated flips in a row. For **n=5** and **k=2**, the sequence **THTTH** is **kGood** because it has no consecutive repeats longer than 2, but the sequence **HHTTT** is not because it ends with 3 tails in a row.

Write the **countKGood** function:

```
int countKGood(int n, int k)
```

The function takes two arguments: **n**, the total number of flips and **k**, the maximum number of consecutive repeats. The return value is the count of length-**n** sequences that are **kGood.**

Specifications:

- Your algorithm must operate by recursive counting.
- You must not create any strings or auxiliary data structures (no vector, map, set, etc).
- You must not generate, store, or print sequences, just count them.
- You will need a helper/wrapper function.
- You may assume that **n >= 1** and **k >= 1.**

Please write your answer on the following page.

```
int countKGood(int n, int k)
```

Problem 5: Recursive backtracking (15 points)
Bored in your chemistry lecture, you look over to the periodic table of the elements and start aimlessly combining atomic symbols to form words: **`He+Al = HeAl, Ga+U+Ge = GaUGe, ReFrIGeRaTiON,`**… You are curious how long a word you can form without repeating a symbol. This is a perfect job for your recursive backtracking skills!

You are to write the **`maxLength`** function:

```
int maxLength(Vector<string>& symbols, Lexicon& lex)
```

This function recursively explores the words that can be formed by combining symbols without repeats and returns the length of the longest such word.

Specifications:
- Your algorithm must be recursive and must use backtracking to generate the result.
- A symbol **cannot be used more than once** when forming a word**.**
- You must use lexicon's **`containsPrefix`** to avoid exploring dead-end paths.
- You will need a helper/wrapper function.
- The function returns only the length of the longest symbol word. Do not print, store, or return the words.

Below we provide the implementation of **`printCombos`** that prints all symbol combinations up to length 4. This is a different task than **`maxLength`** but has some similarities. Reviewing this code may be a useful starting point. You can borrow code and structure from this code as you see fit.

```
void printCombos(Vector<string>& symbols, string soFar) {
    if (soFar.length() > 4) {
        return;
    }
    cout << soFar << endl;
    for (int i = 0; i < symbols.size(); i++) {
        printCombos(symbols, soFar + symbols[i]);
    }
}
```

Example call:

```
Vector<string> symbols = {"H","He","Li","Be","B","C","N","O", ...
printCombos(symbols, "");
```

Here are the first 6 lines of output from the above call. You may find it helpful to trace **`printCombos`** and confirm how it produces this output.

```
             // this is an empty line
H
HH           // printCombos allows symbol re-use (maxLength does not!)
HHH
HHHH         // printCombos stops at length 4 (maxLength does not!)
HHHB         // consider why this is HHHB, not HHHHe
...
```

```
int maxLength(Vector<string>& symbols, Lexicon& lex)
```